Single-generation Network Coding for Networks with Delay

K. Prasad and B. Sundar Rajan

arXiv:0909.1638v1 [cs.IT] 9 Sep 2009

***Abstract*—A single-source network is said to be *memory-free* if all of the internal nodes (those except the source and the sinks) do not employ memory but merely send linear combinations of the incoming symbols (received at their incoming edges) on their outgoing edges. Memory-free networks with delay using network coding are forced to do inter-generation network coding, as a result of which the problem of some or all sinks requiring a large amount of memory for decoding is faced. In this work, we address this problem by utilizing memory elements at the internal nodes of the network also, which results in the reduction of the number of memory elements used at the sinks. We give an algorithm which employs memory at the nodes to achieve single-generation network coding. For fixed latency, our algorithm reduces the total number of memory elements used in the network to achieve single-generation network coding. We also discuss the advantages of employing single-generation network coding together with convolutional network-error correction codes (CNECCs) for networks with unit-delay and illustrate the performance gain of CNECCs by using memory at the intermediate nodes using simulations on an example network under a probabilistic network error model.**

I. Introduction

Network coding was introduced in [1] as a means of achieving maximum rate of transmission in wireline networks. An algebraic formulation of network coding was discussed in [2] for both instantaneous networks and networks with delays. Convolutional network-error correcting codes(CNECCs) were introduced for acyclic instantaneous networks in [3] and for unit-delay, memory-free networks in [4].

In this work, we consider acyclic, single-source networks with delays which have a multicast network code in place. The set of all code symbols generated at the source at any particular time instant is called a *generation*. In unit-delay, memory-free networks, the nodes of the network may receive information of different generations on their incoming edges at every time instant and therefore network coding across generations (*inter-generation*) is unavoidable in general. However, the sinks have to employ memory to decode the symbols. If memory is utilized in the internal nodes also, such inter-generation network coding can be avoided thus making the decoding simpler.

We define a *single-generation network code* as a network code where all the symbols received at all the sinks are linear combinations of the symbols belonging to the same generation. In [5], the technique of adding memory at the nodes to achieve single-generation network coding was discussed. However this was done only on a per-node basis without considering the entire topology or the network code of the network. On the other hand, we consider the entire network topology and the network code, which govern the addition of memory elements at the nodes and the way in which they are rearranged across the network to reduce the overall memory usage in the network.

The organization and contributions of this work are as follows

- After briefly discussing the network setup and the network code for an acyclic network with delays and memory (Section II), we introduce different methods of adding memory at a node and analyze how each of them affect the local and global encoding kernels of the network code (Section III).
- We also present different memory reduction and distribution techniques (Section IV).
- We propose an algorithm which uses the memory at the nodes to achieve single-generation network coding while reducing the overall memory usage in the network (Section V).
- We discuss the advantages of employing memory at the intermediate nodes in tandem with CNECCs in terms of their encoding/decoding (Section VI).
- We illustrate the the performance benefits by using memory for CNECCs for unit-delay networks using simulations on an example unit-delay network under a probabilistic error setting (Section VII).

II. Networks with delay and memory

The model for acyclic networks with delays considered in this paper is as in [2]. An acyclic network can be represented as an acyclic directed multi-graph (a graph that can have parallel edges between nodes) $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ where $\mathcal{V}$ is the set of all vertices and $\mathcal{E}$ is the set of all edges in the network.

We assume that every edge in the directed multi-graph representing the network has unit *capacity* (can carry utmost one symbol from $\mathbb{F}_q$, the field with q elements). Network links with capacities greater than unit are modeled as parallel edges. The network has delays, i.e, every edge in the directed graph representing the input has a unit delay associated with it, represented by the parameter z. Such networks are known as *unit-delay networks*. Those network links with delays greater than unit are modeled as serially concatenated edges in the directed multi-graph. We assume a single-source node $s \in \mathcal{V}$ and a set of sinks $\mathcal{T}$. Let $n_{_T}$ be the unicast capacity for a sink node $T \in \mathcal{T}$ i.e the maximum number of edge-disjoint paths from s to T. Then

$$n_{min} = \min_{T \in \mathcal{T}} n_{_T}$$

is the max-flow min-cut capacity of the multicast connection.

A. Network code for unit-delay, memory-free networks

We follow [2] in describing the network code. For each node $v \in \mathcal{V}$, let the set of all incoming edges be denoted by $\Gamma_I(v)$. Then $|\Gamma_I(v)| = \delta_I(v)$ is the in-degree of v. Similarly the set of all outgoing edges is defined by $\Gamma_O(v)$, and the out-degree of the node v is given by $|\Gamma_O(v)| = \delta_O(v)$.

For any $e \in \mathcal{E}$ and $v \in \mathcal{V}$, let $head(e) = v$, if v is such that $e \in \Gamma_I(v)$. Similarly, let $tail(e) = v$, if v is such that $e \in \Gamma_O(v)$. We will assume an ancestral ordering on $\mathcal{V}$ and $\mathcal{E}$ of the acyclic graph of the unit-delay, memory-free network.

The network code can be defined by the local kernel matrices of size $\delta_I(v) \times \delta_O(v)$ for each node $v \in \mathcal{V}$ with entries from $\mathbb{F}_q$. The global encoding kernels for each edge can be recursively calculated from these local kernels.

The network transfer matrix, which governs the input-output relationship in the network, is defined as given in [2] for an n-dimensional ($n \leq n_{min}$) network code. Towards this end, the matrices A,K,and B^T(for every sink $T \in \mathcal{T}$) are defined as follows.

The entries of the $n \times |\mathcal{E}|$ matrix A are defined as

$$A_{i,j} = \begin{cases} \alpha_{i,e_j} & \text{if } e_j \in \Gamma_O(s) \\ 0 & \text{otherwise} \end{cases}$$

where $\alpha_{i,e_j} \in \mathbb{F}_q$ is the local encoding kernel coefficient at the source coupling input i with edge $e_j \in \Gamma_O(s)$.

The $(i,j)^{th}$ entry of the $|\mathcal{E}| \times |\mathcal{E}|$ matrix K is $K_{e_i,e_j} \in \mathbb{F}_q$ which is the local kernel coefficient between e_i and e_j at the node $head(e_i) = tail(e_j)$ (if such a node exists), and zero if $head(e_i) \neq tail(e_j)$.

For every sink $T \in \mathcal{T}$, the entries of the $|\mathcal{E}| \times n$ matrix B^T are defined as

$$B^T_{i,j} = \begin{cases} \epsilon_{e_j,i} & \text{if } e_j \in \Gamma_I(T) \\ 0 & \text{otherwise} \end{cases}$$

where all $\epsilon_{e_j,i} \in \mathbb{F}_q$.

For unit-delay, memory-free networks, we have

$$F(z) := (I - zK)^{-1}$$

where I is the $|\mathcal{E}| \times |\mathcal{E}|$ identity matrix. Now we have the following definition.

Definition 1 ([2]): The network transfer matrix, $M_T(z)$, corresponding to a sink node $T \in \mathcal{T}$ for a n-dimensional network code, is a full rank (over the field of rationals $\mathbb{F}_q(z)$) $n \times n$ matrix defined as

$$M_T(z) := AF(z)B^T = AF_T(z).$$

With an n-dimensional network code, the input and the output of the network are n-tuples of elements from $\mathbb{F}_q[[z]]$, the formal power series ring over $\mathbb{F}_q$. Definition 1 implies that if $\boldsymbol{x}(z) \in \mathbb{F}_q^n[[z]]$ is the input to the unit-delay, memory-free network, then at any particular sink $T \in \mathcal{T}$, we have the output, $\boldsymbol{y}(z) \in \mathbb{F}_q^n[[z]]$, to be

$$\boldsymbol{y}(z) = \boldsymbol{x}(z)M_T(z).$$

B. Network code for networks with delay and memory

We define the *instantaneous counterpart* of a unit-delay network as follows.

Definition 2: Given a unit-delay network $\mathcal{G}(\mathcal{V},\mathcal{E})$, the network obtained from $\mathcal{G}$ (having the same node set $\mathcal{V}$ and the same edge set $\mathcal{E}$) by removing the delays associated with the edges is defined as the *instantaneous counterpart* of $\mathcal{G}(\mathcal{V},\mathcal{E})$.

Example 1: Fig. 1 illustrates an example. A modified butterfly unit-delay network (top) and its instantaneous counterpart (bottom) are shown. The global kernels of the incoming edges to the sinks T_1 and T_2 corresponding to a 2 dimensional network code are indicated for both networks.

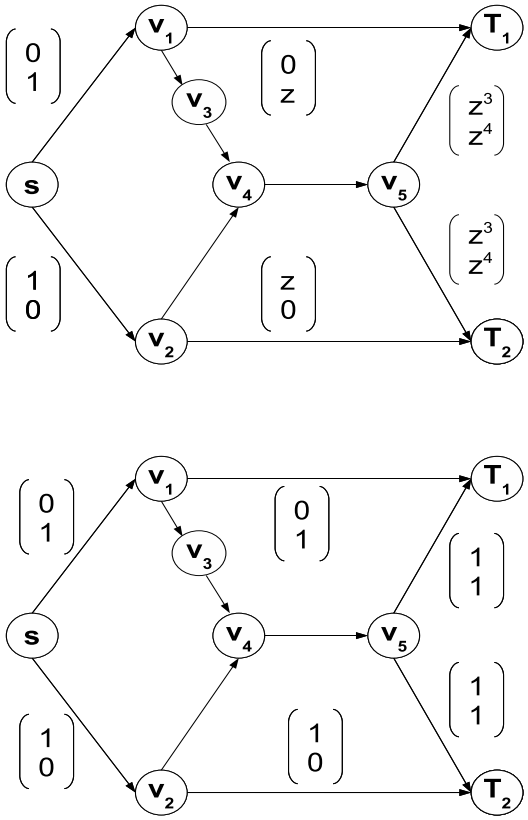

Fig. 1. The figure corresponding to Example 1 (A unit-delay network and its instantaneous counterpart).

Let $\mathcal{G}_m(\mathcal{V},\mathcal{E})$ be a single-source, acyclic network with every edge of the network having some delay (a positive integer) and with memory elements at the nodes available for usage. If none of the memory elements at the nodes are used, then we can model $\mathcal{G}_m$ as a unit-delay, memory-free network $\mathcal{G}_u$. Let $\mathcal{G}_{inst}$ be the instantaneous counterpart of $\mathcal{G}_u$. The following lemma ensures the equivalence of a network code between $\mathcal{G}_{inst}$ and $\mathcal{G}_u$.

Lemma 1 ([4]): Let $\mathcal{G}'(\mathcal{V},\mathcal{E})$ be a single-source acyclic, unit-delay, memory-free network, and $\mathcal{G}'_{inst}$ be the instantaneous counterpart of $\mathcal{G}'$. Let $\mathcal{N}$ be the set of all $\delta_I(v) \times \delta_O(v)$ matrices $\forall\ v \in \mathcal{V}$, i.e, the set of local encoding kernel matrices at each node, describing an m-dimensional network code (over $\mathbb{F}_q$) for $\mathcal{G}'_{inst}$ ($m \leq$ min-cut of the source-sink connections in $\mathcal{G}'_{inst}$). Then the network code described by $\mathcal{N}$ continues to

be an m-dimensional network code (over $\mathbb{F}_q(z)$) for the unit-delay, memory-free network $\mathcal{G}'$.

If the nodes use memory elements such that inter-generation network coding is prevented at any particular node of the network, then this leads to single-generation network coding in the network.

In Section V we give an algorithm which uses memory elements at the nodes to achieve single-generation network coding, i.e, the network transfer matrix $M_T(z)$ of every sink $T \in \mathcal{T}$ in the in $\mathcal{G}_m$ becomes

$$M_T(z) = z^{L_T} M_T \tag{1}$$

where L_T is some positive integer and M_T is the network transfer matrix of the sink T in $\mathcal{G}_{inst}$. Clearly, if M_T is full rank (over $\mathbb{F}_q$), so is $M_T(z)$ (over $\mathbb{F}_q(z)$).

III. Memory additions at a node

For the source node s, let $\tilde{\Gamma}_I(s)$ denote the set of n virtual incoming edges which denote the n inputs. The global kernels of these edges are therefore the columns of an $n \times n$ identity matrix over $\mathbb{F}_q$, the field over which the network code is defined. For every non-source node $v \in \mathcal{V}$, let $\tilde{\Gamma}_I(v) = \phi$. For a sink $T \in \mathcal{T}$, let $\tilde{\Gamma}_O(T)$ denote n virtual outgoing edges denoting the n outputs at sink T. The global kernels of these edges are the columns of the network transfer matrix $M_T(z)$. For every non-sink node $v \in \mathcal{V}$, let $\tilde{\Gamma}_O(v) = \phi$. We then define the set $\tilde{\mathcal{E}}$ as

$$\tilde{\mathcal{E}} := \mathcal{E} \cup \tilde{\Gamma}_I(s) \cup \left(\bigcup_{T \in \mathcal{T}} \tilde{\Gamma}_O(T) \right)$$

The ancestral ordering on $\mathcal{E}$ can then be extended to an ancestral ordering on $\tilde{\mathcal{E}}$.

For any $e_i, e_j \in \tilde{\mathcal{E}}$ such that $head(e_i) = tail(e_j) = v \in \mathcal{V}$, with memory being used at v, the local kernel A_{e_i,e_j} (the kernel coefficient between $e_i \in \tilde{\Gamma}_I(s)$ and $e_j \in \Gamma_O(s)$ with $s = v$), K_{e_i,e_j} or $B^v_{e_i,e_j}$ (the kernel coefficient between e_i and $e_j \in \tilde{\Gamma}_O(v)$ for some sink node v) can have elements from $\mathbb{F}_q(z)$. We show in Section V that using the memory elements at the nodes according to Subsection III-A and Subsection III-B is sufficient to guarantee single-generation network coding at each node and therefore in the given network.

A. Adding memory at a node for a pair of an incoming and an outgoing edge

For any $e_i, e_j \in \tilde{\mathcal{E}}$ such that $head(e_i) = tail(e_j) = v \in \mathcal{V}$, we define M_{e_i,e_j} as the number of memory elements utilized at the node v to delay the symbols coming from the incoming edge e_i (before any network coding is performed at node v on the symbols from e_i) such that the local kernel between e_i and e_j is modified in one of the following ways

$$A_{e_i,e_j} \longmapsto z^{M_{e_i,e_j}} A_{e_i,e_j} \quad \text{if } e_i \in \tilde{\Gamma}_I(s), e_j \in \mathcal{E} \tag{2}$$

$$K_{e_i,e_j} \longmapsto z^{M_{e_i,e_j}} K_{e_i,e_j} \quad \text{if } e_i, e_j \in \mathcal{E} \tag{3}$$

$$B^v_{e_i,e_j} \longmapsto z^{M_{e_i,e_j}} B^v_{e_i,e_j} \quad \text{if } e_i \in \mathcal{E}, e_j \in \tilde{\Gamma}_O(v) \tag{4}$$

while none of the other local kernels are changed. The matrix $F(z) = (I - zK)^{-1}$ is also correspondingly modified.

B. Adding memory at a node for an outgoing edge

For $e_j \in \Gamma_O(v) \cup \tilde{\Gamma}_O(v)$, we define $M_{e_j,tail(e_j)}$ as the number of memory elements added at node v to delay the symbols going into the edge e_j after performing network coding at v. In such a case, the elements of the matrix K (or of the matrix or A, or B^v) are modified according to the following rule.

$$A_{e_i,e_j} \longmapsto z^{M_{e_j,tail(e_j)}} A_{e_i,e_j} \quad \forall e_i \in \Gamma_{I,e_j}(v), \text{ if } v = s \tag{5}$$

$$K_{e_i,e_j} \longmapsto z^{M_{e_j,tail(e_j)}} K_{e_i,e_j} \quad \forall e_i \in \Gamma_{I,e_j}(v), \text{ if } v \neq s, e_j \in \Gamma_O(v) \tag{6}$$

$$B^v_{e_i,e_j} \longmapsto z^{M_{e_j,tail(e_j)}} B^v_{e_i,e_j} \quad \forall e_i \in \Gamma_{I,e_j}(v), \text{if } e_j \in \tilde{\Gamma}_O(v) \tag{7}$$

where the set $\Gamma_{I,e_j}(v) \subseteq \Gamma_I(v) \cup \tilde{\Gamma}_I(v)$ is defined as in the top of the next page. The elements of the matrix $F(z)$ are also correspondingly modified.

Example 2: Fig 2 illustrates an example of the memory additions at a node. The memory elements indicated inside the box labeled 'A' are added at the node for the pair of edges e_i and e_j thereby delaying the symbols on e_i before network coding at the node, i.e, $M_{e_i,e_j} = 2$. Similarly the memory element indicated by 'C' is added for the pair of edges e_i and e_k, i.e, $M_{e_i,e_k} = 1$. The memory element indicated by 'B' is added for the outgoing edge e_j after network coding, i.e, $M_{e_j,tail(e_j)} = 1$.

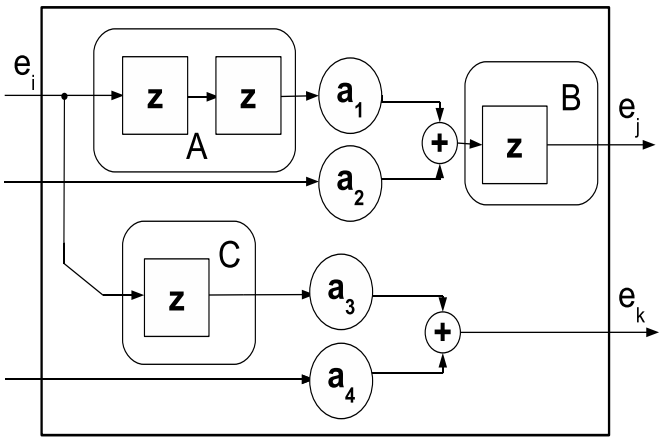

Fig. 2. The figure corresponding to Example 2 (Adding memory at a node).

IV. Memory reduction and distribution techniques

In this section, we look at techniques to reduce the memory used at the nodes of the network and the overall memory used in the network and also to obtain a fairly uniform memory usage distribution throughout the network.

We define the maximum number of memory elements added to delay the symbols coming from an edge $e_i \in \tilde{\mathcal{E}}$ into node $head(e_i) = v$ as

$$M_{e_i,head(e_i),max} := \max_{e_j \in \Gamma_{O,e_i}(v)} Me_i, e_j \tag{9}$$

$$\Gamma_{I,e_j}(v) := \left\{e_i \in \Gamma_I(v) \mid K_{e_i,e_j} \neq 0\right\} \bigcup \left\{e_i \in \tilde{\Gamma}_I(v) \mid A_{e_i,e_j} \neq 0\right\}. \tag{8}$$

$$\Gamma_{O,e_i}(v) := \left\{e_j \in \Gamma_O(v) \mid K_{e_i,e_j} \neq 0\right\} \bigcup \left\{e_j \in \tilde{\Gamma}_O(v) \mid B^v_{e_i,e_j} \neq 0\right\} \tag{10}$$

where $\Gamma_{O,e_i}(v)$ is defined as shown at the top of the next page. We define the total number of memory elements used at node v as

$$M_v = \sum_{e_i \in \Gamma_I(v) \cup \tilde{\Gamma}_I(v)} M_{e_i,head(e_i),max} + \sum_{e_j \in \Gamma_O(v) \cup \tilde{\Gamma}_O(v)} M_{e_j,tail(e_j)}.$$

A. Memory reduction in a single node

Consider a node $v \in \mathcal{V}$ in which memory elements have been added to delay symbols coming from an edge $e_i \in \Gamma_I(v) \cup \tilde{\Gamma}_I(v)$.

Then, retaining the $M_{e_i,head(e_i),max}$(as defined in (9)) memory elements, all other memory elements placed on e_i can be removed without any change in any local or global kernels by tapping symbols from the $M_{e_i,head(e_i),max}$ memory elements wherever necessary. Doing this for every incoming edge of v is equivalent to obtaining a minimal encoder (one with minimum number of memory elements) of the transfer function (input-output relationship) at node v.

Example 3: Fig. 3 illustrates a particular example of such a reduction. The figure on the top (all $a_i \in \mathbb{F}_q$) represents a node v before memory reduction with $M_v = 3$, while the figure on the bottom is the same node after memory reduction with $M_v = 2$.

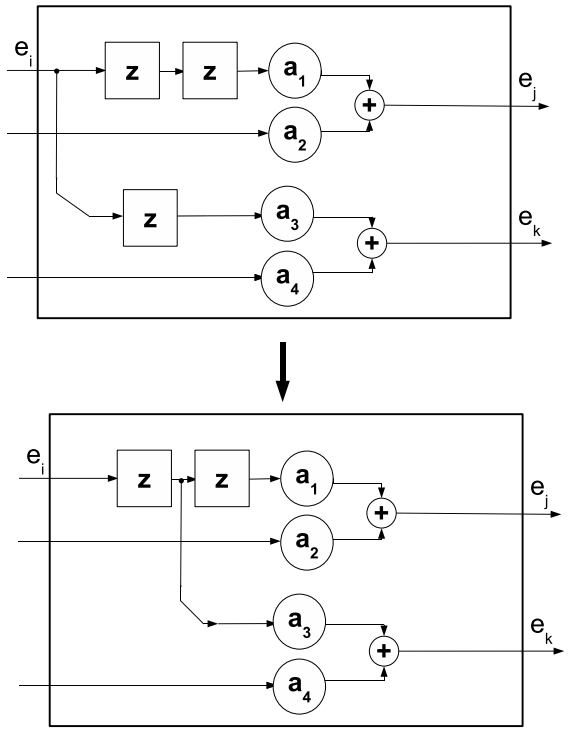

Fig. 3. The figure corresponding to Example 3 (Memory reduction at a node).

B. Memory reduction between nodes

For a set of edges $\mathcal{E}' \subseteq \tilde{\mathcal{E}}$, let $\mathcal{V}_{\mathcal{E}'}$ be the set of all nodes defined as follows

$$\mathcal{V}_{\mathcal{E}'} = \{head(e_j) \mid e_j \in \mathcal{E}'\} \tag{11}$$

We now define $M_{e_i,head(e_i),min}$ and $M_{\mathcal{E}'}$ as follows.

$$M_{e_i,head(e_i),min} := \min_{e_j \in \Gamma_{O,e_i}(v)} Me_i, e_j \tag{12}$$

$$M_{\mathcal{E}'} := \min_{e_j \in \mathcal{E}'} M_{e_j,head(e_j),min} \tag{13}$$

where $\Gamma_{O,e_i}(v)$ is as defined in (10).

For a node $v \in \mathcal{V}$, we define the set of *adjacent nodes* of v as the set of nodes

$$\mathcal{E}_v := \{v' \mid v' = head(e_j)\ \forall e_j \in \Gamma_O(v)\}.$$

1) Memory reduction between adjacent nodes: For a node $v \in \mathcal{V}$, and for some $\Gamma'_O(v) \subseteq \Gamma_O(v) \cup \tilde{\Gamma}_O(v)$, let $\Gamma'_I(v) \subseteq \Gamma_I(v) \cup \tilde{\Gamma}_I(v)$ be defined as

$$\Gamma'_I(v) = \bigcup_{e_j \in \Gamma'_O(v)} \Gamma_{I,e_j}(v).$$

where $\Gamma_{I,e_j}(v)$ is as in (8), i.e, the global kernels of the edges in $e_j \in \Gamma'_O(v)$ are linear combinations of the global kernels of the edges in $\Gamma'_I(v)$ only and none else. Also let $M_{\Gamma'_O(v)}$ and the set $\mathcal{V}_{\Gamma'_O(v)} \subseteq \mathcal{E}_v$ of nodes be defined for the set of edges $\Gamma'_O(v)$ as in (13) and (11) respectively.

We define the term $M_{e_i,\Gamma'_O(v)}$ as

$$M_{e_i,\Gamma'_O(v)} = \max\left\{0, M_{\Gamma'_O(v)} - M_{e_i,head(e_i),max}\right\} \tag{14}$$

Then, if the condition is satisfied,

$$\sum_{e_i \in \Gamma'_I(v)} M_{e_i,\Gamma'_O(v)} \leq M_{\Gamma'_O(v)} |\Gamma'_O(v)| \tag{15}$$

then all of the $|\Gamma'_O(v)| M_{\Gamma'_O(v)}$ used at the nodes $\mathcal{V}_{\Gamma'_O(v)}$ (to delay symbols coming from the edges $e_j \in \Gamma'_O(v)$) can be 'absorbed' into node v by removing all these memory elements and adding $M_{e_i,\Gamma'_O(v)}$ memory elements at node v for every $e_i \in \Gamma'_I(v)$ (and thereby used for delaying the symbols coming from every $e_i \in \Gamma'_I(v)$), without using any additional memory and without changing the global kernels of any outgoing edge of any node in $\mathcal{V}_{\Gamma'_O(v)}$.

This technique of 'absorption' of the memory elements from a set of nodes which are the 'heads' of the outgoing edges from a node v, to the node v itself, is beneficial in terms of reducing the overall memory usage of the network (to achieve single-

generation network coding) if the condition (15) is satisfied as a strict inequality.

Example 4: Fig. 4 illustrates an example for memory reduction between multiple nodes (v_1, v_2, v_3 and v_4 here) of a network. Here $M_{\Gamma'_O(v)} = 1, |\Gamma'_O(v)| = 3$, and $M_{e_1,\Gamma'_O(v)} = M_{e_2,\Gamma'_O(v)} = 1$. Therefore, three memory elements at nodes v_2, v_3 and v_4 are 'absorbed' into two memory elements at node v_1. The boxes indicate the use of memory elements and the node to which the memory elements are attached.

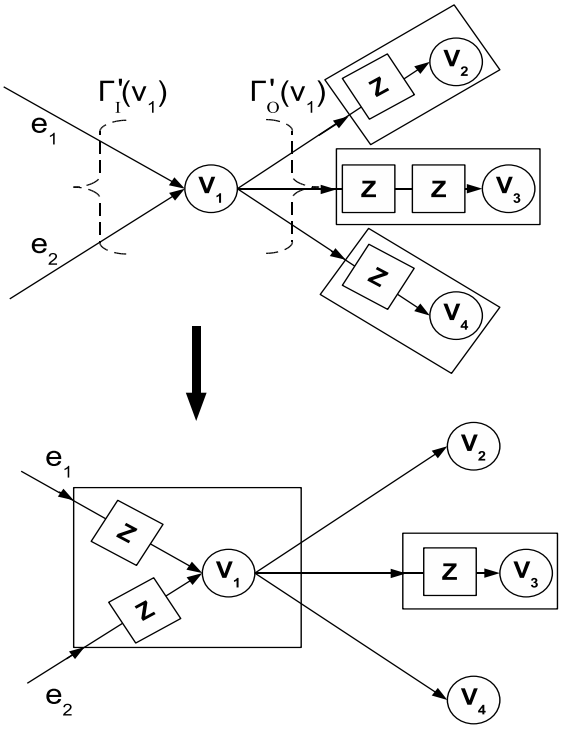

Fig. 4. The figure corresponding to Example 4 (Memory reduction between adjacent nodes).

2) Memory reduction between nodes not necessarily adjacent: For $\mathcal{E}_I, \mathcal{E}_O \subset \tilde{\mathcal{E}}$ being two sets of edges, we say that they form a pair $[\mathcal{E}_I, \mathcal{E}_O]$ if

$$\mathcal{E}_I = \bigcup_{e_j \in \mathcal{E}_O} \Gamma_{I,e_j}(tail(e_j)).$$

and

$$\mathcal{E}_O = \bigcup_{e_i \in \mathcal{E}_I} \Gamma_{O,e_i}(head(e_i)).$$

We say that the sets $\mathcal{E}_I, \mathcal{E}_O$ form a pair $[\mathcal{E}_I, \mathcal{E}_O)$ if

$$\mathcal{E}_I = \bigcup_{e_j \in \mathcal{E}_O} \Gamma_{I,e_j}(tail(e_j)).$$

and

$$\mathcal{E}_O \subset \bigcup_{e_i \in \mathcal{E}_I} \Gamma_{O,e_i}(head(e_i)).$$

For a node v, we define the set P_v as follows

$$P_v := \{[\Gamma_{I_i}(v), \Gamma_{O_i}(v)] \mid 1 \leq i \leq s_v\}$$

such that the following conditions are satisfied

$$\Gamma_{I_i}(v) \cap \Gamma_{I_j}(v) = \phi, \ \forall\ 1 \leq i, j \leq s_v, \ i \neq j \quad (16)$$

$$\Gamma_{O_i}(v) \cap \Gamma_{O_j}(v) = \phi, \ \forall\ 1 \leq i, j \leq s_v, \ i \neq j \quad (17)$$

where s_v is the maximum number of sets satisfying conditions (16) and (17). Algorithm 1 shown at the top of the next page obtains the set P_v for some node v.

Example 5: Fig. 5 illustrates a node v with the local kernel matrix over some field $\mathbb{F}_q$. For this node, the set P_v is given as

$$P_v = \{[\Gamma_{I_1}(v), \Gamma_{O_1}(v)], [\Gamma_{I_2}(v), \Gamma_{O_2}(v)]\}$$

where

$$\Gamma_{I_1}(v) = \{e_1, e_2, e_3\} \qquad \Gamma_{O_1}(v) = \{e_5\}$$
$$\Gamma_{I_2}(v) = \{e_4\} \qquad \Gamma_{O_2}(v) = \{e_6, e_7, e_8\}.$$

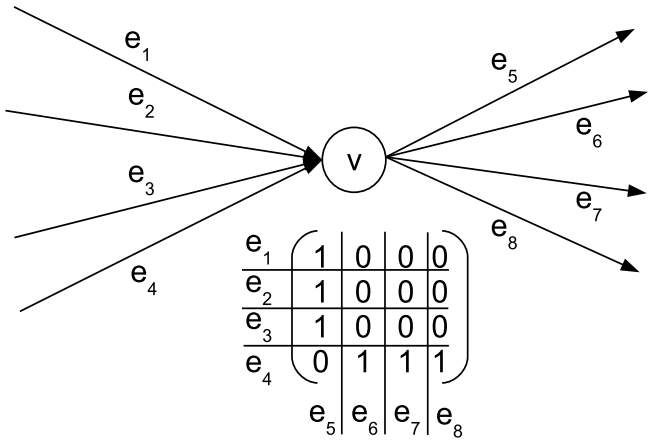

Fig. 5. The figure corresponding to Example 5 which gives the set P_v of the node v.

For an pair of edge-sets $[\Gamma_{I_i}(v), \Gamma_{O_i}(v)] \in P_v$, we define $S_i(v)$, a sequence of pairs of edge-sets as

$$S_i(v) := \left[\mathcal{E}_{i_m}, \mathcal{E}_{i_{m-1}}\right), \left[\mathcal{E}_{i_{m-1}}, \mathcal{E}_{i_{m-2}}\right], ..., [\mathcal{E}_{i_2}, \mathcal{E}_{i_1}], [\mathcal{E}_{i_1}, \mathcal{E}_{o_1}] \quad (18)$$

where $[\mathcal{E}_{i_1}, \mathcal{E}_{o_1}] = [\Gamma_{I_i}(v), \Gamma_{O_i}(v)]$, and m is the maximum length of the sequence, that is possible to be obtained as in (18) for the edge-set pair $[\Gamma_{I_i}(v), \Gamma_{O_i}(v)]$.

Let k be an integer such that

$$|\mathcal{E}_{i_k}| = \min_{1 \leq j \leq m} |\mathcal{E}_{i_j}|.$$

For the set $\Gamma_{O_i}(v)$, let $M_{\Gamma_{O_i}}(v)$ be defined as in (13), and the set of nodes $\mathcal{V}_{\Gamma_{O_i}(v)}$ be defined as in (11). Let the set of nodes $\mathcal{V}_{\mathcal{E}_{i_k}}$ be defined as in (11) for the set $\mathcal{E}_{i_k}$. Also, let $M_{e_{i_k},\Gamma_{O_i}}(v)$ be defined as in (14) for the set $\Gamma_{O_i}(v)$ and for an edge $e_{i_k} \in \mathcal{E}_{i_k}$. As in the memory reduction procedure of adjacent nodes, if

$$\sum_{e_{i_k} \in \mathcal{E}_{i_k}} M_{e_{i_k},\Gamma_{O_i}(v)} \leq M_{\Gamma_{O_i}}(v)|\Gamma_{O_i}(v)| \quad (19)$$

then the $|\Gamma_{O,i}(v)|M_{\Gamma_{O,i}(v)}$ used at the nodes $\mathcal{V}_{\Gamma_{O_i}(v)}$ (to delay symbols coming from the edges $e_j \in \Gamma_{O,i}(v)$) can be

Input: A node $v \in \mathcal{V}$ with the edge sets $\Gamma_I(v) \cup \tilde{\Gamma}_I(v)$ and $\Gamma_O(v) \cup \tilde{\Gamma}_O(v)$.
Output: The set P_v for the node v.

Let $i = 1, Out(v) = \Gamma_O(v) \cup \tilde{\Gamma}_O(v), P_v = \phi$.
repeat
  Let $\Gamma_{I_i}(v) = \Gamma_{O_i}(v) = \phi$.
  For some $e_j \in Out(v)$, let $\Gamma_{I_i}(v) = \Gamma_{I,e_j}(v)$
  repeat
    Let
$$\Gamma_{O_i}(v) = \bigcup_{e_i \in \Gamma_{I_i}(v)} \Gamma_{O,e_i}(v)$$
    Let
$$\Gamma_{I_i}(v) = \bigcup_{e_j \in \Gamma_{O_i}(v)} \Gamma_{I,e_j}(v)$$
  until *the sets* $\Gamma_{I_i}(v)$ *and* $\Gamma_{O_i}(v)$ *remain unchanged for* 2 *consecutive iterations* ;
  Let $P_v = P_v \cup \{[\Gamma_{I_i}(v), \Gamma_{O_i}(v)]\}$.
  Let $Out(v) = Out(v) \backslash \Gamma_{O_i}(v)$ and $i = i + 1$.
until $Out(v) = \phi$;

Algorithm 1. Algorithm to obtain the set P_v for a node v.

removed without changing the global kernels of the edges of $\Gamma_O(v')$, $\forall\ v' \in \mathcal{V}_{\Gamma_{O_i}(v)}$ by adding $M_{e_{i_k},\Gamma_{O_i}(v)}$ memory elements for each edge $e_{i_k} \in \mathcal{E}_{i_k}$ at the node $head(e_{i_k}) \in \mathcal{V}_{\mathcal{E}_{i_k}}$. This technique will save memory if the condition (19) is satisfied as a strict inequality.

Example 6: Figure 6 illustrates an example for the memory reduction procedure between non-adjacent nodes. Let $K_{e_i,e_j} \neq 0$, $\forall\ 9 \leq i \leq 12$, $13 \leq j \leq 15$. In the example, for the node v_3, the set P_{v_3} and the sequence $S_1(v_3)$ corresponding to the only element of P_{v_3} are given by (20) and (21) at the top of the next page.

Now, we have $M_{\Gamma_{O,1}(v_3)} = 1$, $|\Gamma_{O,1}(v_3)| = 3$, $\mathcal{E}_{i_k} = \{e_1\}$ and $M_{e_1,\Gamma_{O,1}(v_3)} = 1$. Therefore, the 3 memory used for the edges in $\Gamma_{O,1}(v_3)$ at the nodes v_4, v_5, and v_6 are 'absorbed' into a single memory element used at node v_1 for edge e_1, thus reducing the memory usage by 2.

Remark 1: The memory reduction procedures of Subsubsection IV-B1, and Subsubsection IV-B2 can sometimes result in exactly the same memory reduction event. However, there could be instances in which only one of the procedures can achieve memory reduction.

For example, the memory reduction procedure of Subsubsection IV-B1 cannot reduce memory at node v_3 in the situation shown in Example 6 because for any $\Gamma'_O(v) \subseteq \Gamma_O(v)$, $|\Gamma'_I(v)| > 3 \geq |\Gamma'_O(v)|$, since $\Gamma'_I(v) = \Gamma_I(v)$. However the memory reduction procedure of Subsubsection IV-B2 does work as shown in Fig 6.

Similarly, in some cases, at a node, the procedure of Subsubsection IV-B1 can be used to reduce memory usage, while Subsubsection IV-B2 cannot be applied. This is because of the fact that, at any node, the procedure of Subsubsection IV-B2 takes into account only those sets of the form P_v, while the procedure of Subsubsection IV-B1 takes into account all possible incoming and outgoing edges. Such a case is seen in Example 7.

Example 7: Fig. 7 shows the node v of Fig. 5 (Example 5) in a particular configuration. The memory reduction procedure of Subsubsection IV-B2 cannot be applied for the set $\Gamma_{O,2}(v)$ because $M_{\Gamma_{O,2}(v)} = 0$.

But $M_{\{e_6,e_7\}} = 1$, and therefore 2 memory elements at node v_1 and v_2 can be absorbed into a single memory element at node v, thereby facilitating memory reduction according to Subsubsection IV-B1.

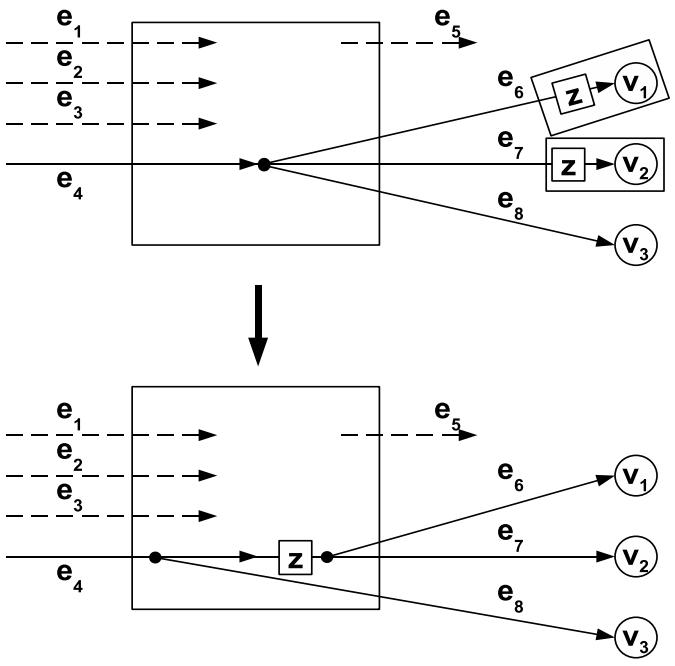

Fig. 7. The figure corresponding to Example 7. The box with the incoming edges e_1, e_2, e_3, and e_4 represents the node v of Fig. 5 (Example 5).

$$P_{v_3} = \{\ [\Gamma_{I_1}(v_3) = \{e_9, e_{10}, e_{11}, e_{12}\}, \Gamma_{O_1}(v_3) = \{e_{13}, e_{14}, e_{15}\}]\ \}. \tag{20}$$

$$S_1(v_3) = [\{e_1\}, \{e_2, e_3\}),\ \ [\{e_2, e_3\}, \{e_5, e_6, e_7, e_8\}],\ \ [\{e_5, e_6, e_7, e_8\}, \Gamma_{I_1}(v_3)],\ \ [\Gamma_{I_1}(v_3), \Gamma_{O_1}(v_3)] \tag{21}$$

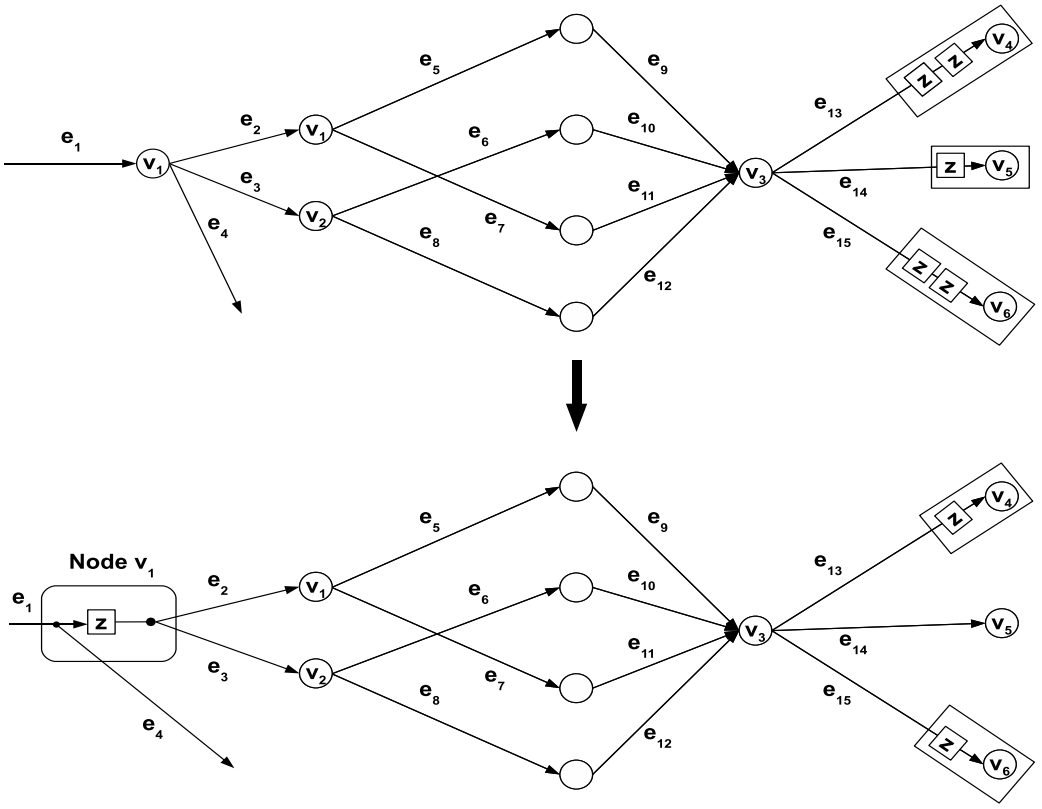

Fig. 6. The figure corresponding to Example 6 (Memory reduction between non-adjacent nodes).

C. Memory distribution

The following technique can be used to distribute memory elements throughout the network in a somewhat uniform way. Suppose there exists a node $v \in \mathcal{V}$ such that for some $e_j \in \Gamma_O(v)$ with $v' = head(e_j)$ and for some integer $m \leq M_{e_j, head(e_j), min}$,

$$M_v + m \leq M_{v'} - m \tag{22}$$

then the m memory elements at node v' used to delay symbols coming from edge e_j can be 'absorbed' into node v (thereby using them to delay symbols going into edge e_j) without changing the global kernels of any edge in $\Gamma_O(v')$.

This technique reduces the number of memory elements used at node v' for delaying its incoming symbols while increasing the number ($M_{e_j, tail(e_j)}$) of memory elements used at node v for delaying its outgoing symbols.

Example 8: Fig 8 illustrates an example for memory distribution between two nodes v_1 and v_2. In the figure on the top, $m = 1, M_{v_1} = 0$, and $M_{v_2} = 3$. Therefore one memory element from v_2 (used to delay symbols coming from e_j into v_2) can be 'absorbed' into node v_1 (and thereby used to delay symbols going into e_j from v_1). The boxes indicate the node to which the memory elements are attached. After distribution, $M_{v_1} = 1$, and $M_{v_2} = 2$.

V. Single-generation network coding - Algorithm

This section presents the main contribution of this paper.

For an edge $e_i \in \mathcal{E}$, let $\boldsymbol{f}_{e_i}(z) \in \mathbb{F}_q^n(z)$ represent the global kernel of e_i. We say that a node $v \in \mathcal{V} \setminus \{s\}$ is a *coding node* if the global kernel of at least one of its outgoing edge is a $\mathbb{F}_q(z)$ linear combination of the global kernels of at least two of its incoming edges. Otherwise, we call v a *forwarding node*.

Let $\mathcal{V}_{cod}$ be the set of coding nodes, and $\mathcal{V}_{fwd}$ be the set of forwarding nodes. Let $\mathcal{V}_{cod}^0$ be the set of all coding nodes such that there exist no path in the network from any other coding node to any node in $\mathcal{V}_{cod}^0$.

Towards proposing an algorithm to enable single-generation network coding, we make some observations and discuss the addition of memory elements at the coding nodes to achieve single-generation network coding.

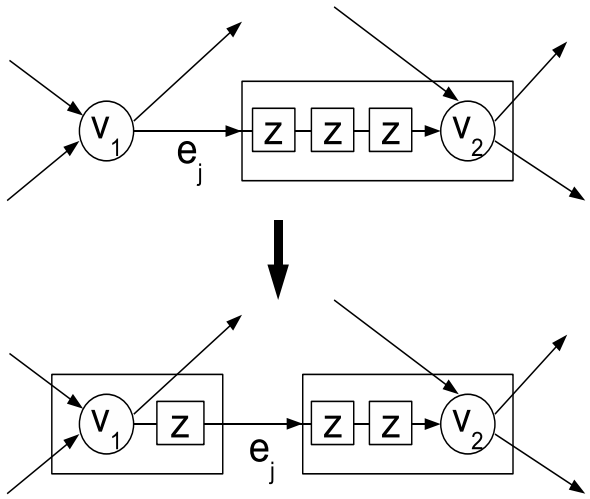

Fig. 8. The figure corresponding to Example 8 (Memory distribution).

Observation 1: For any $v \in \mathcal{V}_{cod}^0$, the global kernel of any $e \in \Gamma_I(v)$ is of the form

$$\boldsymbol{f}_e(z) = z^{l_e}\boldsymbol{f}_e \tag{23}$$

for some positive integer l_e, with $\boldsymbol{f}_e \in \mathbb{F}_q^n$. If the network is a unit-delay network and the node v uses no memory, the global kernel of any $e_j \in \Gamma_O(v)$ is of the form

$$\boldsymbol{f}_{e_j}(z) = \sum_{e_i \in \Gamma_I(v)} zK_{e_i,e_j}\boldsymbol{f}_{e_i}(z) = \sum_{e_i \in \Gamma_I(v)} K_{e_i,e_j} z^{l_{e_i}+1}\boldsymbol{f}_{e_i} \tag{24}$$

where l_{e_i} is a positive integer signifying accumulated delay from the source to edge e_i, and $K_{e_i,e_j} \in \mathbb{F}_q$ signifies the local kernel coefficient between e_i and e_j. The additional z is to account for the delay in the unit delay network.

A. Single-generation processing at the nodes

For every pair of edges $e_i, e_{i'} \in \Gamma_{I,e_j}(v)$ ($\Gamma_{I,e_j}(v)$ being as in (8)) in (24) such that $l_{e_i} < l_{e_{i'}}$, we may add $M_{e_i,e_j} = l_{e_{i'}} - l_{e_i}$ memory elements at node v to delay the symbols coming from e_i such that the global kernel of the edge e_j becomes

$$\boldsymbol{f}_{e_j}(z) = z^{l_{e_j,max}+1} \sum_{e_i \in \Gamma_I(v)} K_{e_i,e_j}\boldsymbol{f}_{e_i} \tag{25}$$

where $l_{e_j,max} = \max_{e_i \in \Gamma_{I,e_j}(v)} l_{e_i}$ and $K_{e_i,e_j} \in \mathbb{F}_q$. Once this process of using memory at the node v results in the global kernel of every edge in $\Gamma_O(v)$ to be a linear combination of symbols from the same generation (generations between different outgoing edges need not be the same), we say that *single-generation processing* has been achieved at node v. For a node $T \in \mathcal{T}$, we say that single-generation processing has been achieved at sink T if the condition (1) is satisfied along with condition (25) for each $e_j \in \Gamma_O(T)$.

Observation 2: We iteratively define the set $\mathcal{V}_{cod}^i \subseteq \mathcal{V}_{cod}$ as the set of coding nodes which have path only from

$$\left(\bigcup_{j=0}^{i-1} \mathcal{V}_{cod}^j\right) \bigcup \mathcal{V}_{fwd}$$

where $\mathcal{V}_{cod}^0$ is as defined before. Once memory has been used to achieve single-generation processing at all nodes in $\mathcal{V}_{cod}^{i-1}$, it can be observed that the global kernels of the incoming and outgoing edges of any node $v \in \mathcal{V}_{cod}^i$ satisfy the same condition as in (23) and (24).

Thus again memory elements can be used at the nodes of $\mathcal{V}_{cod}^i$ to implement single-generation processing, ultimately achieving single-generation processing at each coding node of the network.

B. Algorithm for single-generation network coding

Algorithm 2 shown in the next page is used to achieve single-generation network coding using memory at the nodes of the network, while trying to minimize the total number of memory elements used in the network.

Remark 2: Algorithm 2 assumes that every node has unlimited memory to use and then tries to obtain a configuration that reduces the number of memory elements used in the network. However, if the maximum available memory in the nodes is limited, then the following techniques may be adopted after running Algorithm 2.

- In line 27 of the algorithm, instead of checking condition (22) at every pair of nodes connected by some edge, the actual memory capability of the nodes must be taken into account and then the distribution procedure of Subsection IV-C can be run.
- Finally, at every node in which the algorithm demands more memory elements than what is available, sufficient memory elements should be removed so that the total memory used at the node is utmost what is available. As the penalty of removing these memory elements will be compensated by the sinks, the memory elements that will be removed at the nodes should ideally be such that the compensation occurs in the least number of sinks in the least possible quantity.

Example 9: Fig. 10, Fig. 11, and Fig. 12 represent the network at various stages of the algorithm applied on a modified double-butterfly network as shown in Fig. 9. The modified unit-delay double-butterfly network shown in Fig. 10 has the standard network code over $\mathbb{F}_2$. s is the source node, $T_i, i = 1, 2, 3, 4$ are the sinks. The dotted lines represent the virtual input edges at the source and virtual output edges at the sinks.

Table I shows the network transfer matrices before and after obtaining single-generation processing using Algorithm 2. Table I also shows a comparison between the memory requirements at the sinks (for decoding) between inter-generation network coding (i.e the memory-free case; the numbers shown are the sum of the row degrees of realizable inverse matrices in the third column) and single-generation network coding (as shown in Fig. 12). In the memory-free case, assuming that

Input: A network $\mathcal{G}_m$ with delays and unused memory elements

Output: The network $\mathcal{G}_m$ with a single-generation network code using memory elements at nodes

foreach $v \in \mathcal{V}_{cod}$ *in the ancestral order* **do**
  Introduce sufficient memory elements at node v accordingly as in Subsection V-A in order to enable single-generation processing at node v.
  foreach $e_i \in \Gamma_I(v) \cup \tilde{\Gamma}_I(v)$ **do**
    Run the memory reduction procedure as in Subsection IV-A.
  end
end

Now the global kernel of any edge $e_j \in \Gamma_I(T)$ of any sink T is of the form

$$\boldsymbol{f}_{e_j}(z) = z^{L_{e_j}} \boldsymbol{f}_{e_j}$$

for some positive integer L_{e_j}, with $\boldsymbol{f}_{e_j} \in \mathbb{F}_q^n$.

foreach $T \in \mathcal{T}$ **do**
  Add sufficient memory according to Subsection III-A and Subsection III-B such that single-generation processing is achieved at the sink T.
end
foreach $v \in \mathcal{V}$ *in the reverse-ancestral order* **do**
  foreach *pair of edge-sets* $[\Gamma_{I_i}(v), \Gamma_{O_i}(v)] \in P_v$ **do**
    if *condition (19) is satisfied* **then**
      Run the memory reduction procedure as in Subsubsection IV-B2.
    end
  end
end
foreach $v \in \mathcal{V}$ *in the reverse-ancestral order* **do**
  foreach *subset* $\Gamma'_O(v) \subseteq \Gamma_O(v) \cup \tilde{\Gamma}_O(v)$ **do**
    if *condition (15) is satisfied* **then**
      Run the memory reduction procedure as in Subsubsection IV-B1.
    end
  end
end
foreach $v \in \mathcal{V}$ *in the ancestral order* **do**
  foreach $e_j \in \Gamma_O(v)$ **do**
    if *condition (22) is satisfied* **then**
      Run the memory distribution procedure at v as in Subsection IV-C.
    end
  end
end
foreach $v \in \mathcal{V}$ *in the ancestral order* **do**
  foreach $e_j \in \Gamma_O(v) \cup \tilde{\Gamma}_O(v)$ **do**
    foreach $e_i \in \Gamma_{I,e_j}(v)$ **do**
      Update the corresponding elements in A, K, and B^v matrices according to (2), (3), and (4) of Subsection III-A upon calculating M_{e_i,e_j}.
    end
    Update the corresponding elements in A, K, and B^v matrices according to (5), (6), and (7) of Subsection III-B upon calculating $M_{e_j,tail(e_j)}$.
  end
end

Algorithm 2. Algorithm for using memory at nodes to obtain a single-generation network code

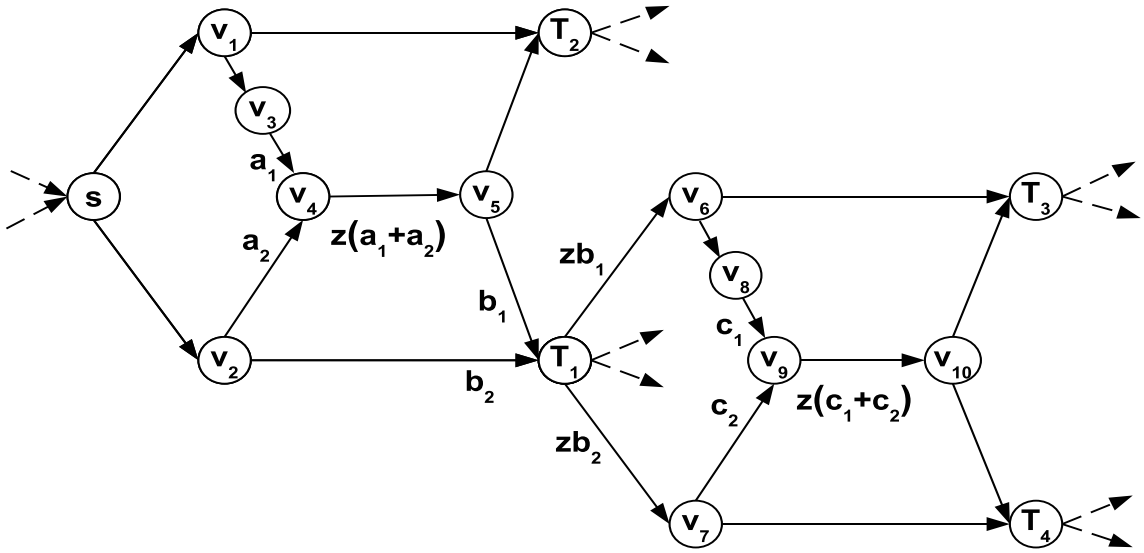

Fig. 9. Figure corresponding to Example 9. A modified double-butterfly network. The mapping between the incoming and outgoing symbols ($a_1, a_2, b_1, b_2, c_1, c_2 \in \mathbb{F}_2$) at the nodes v_4, T_1, and v_9 are shown.

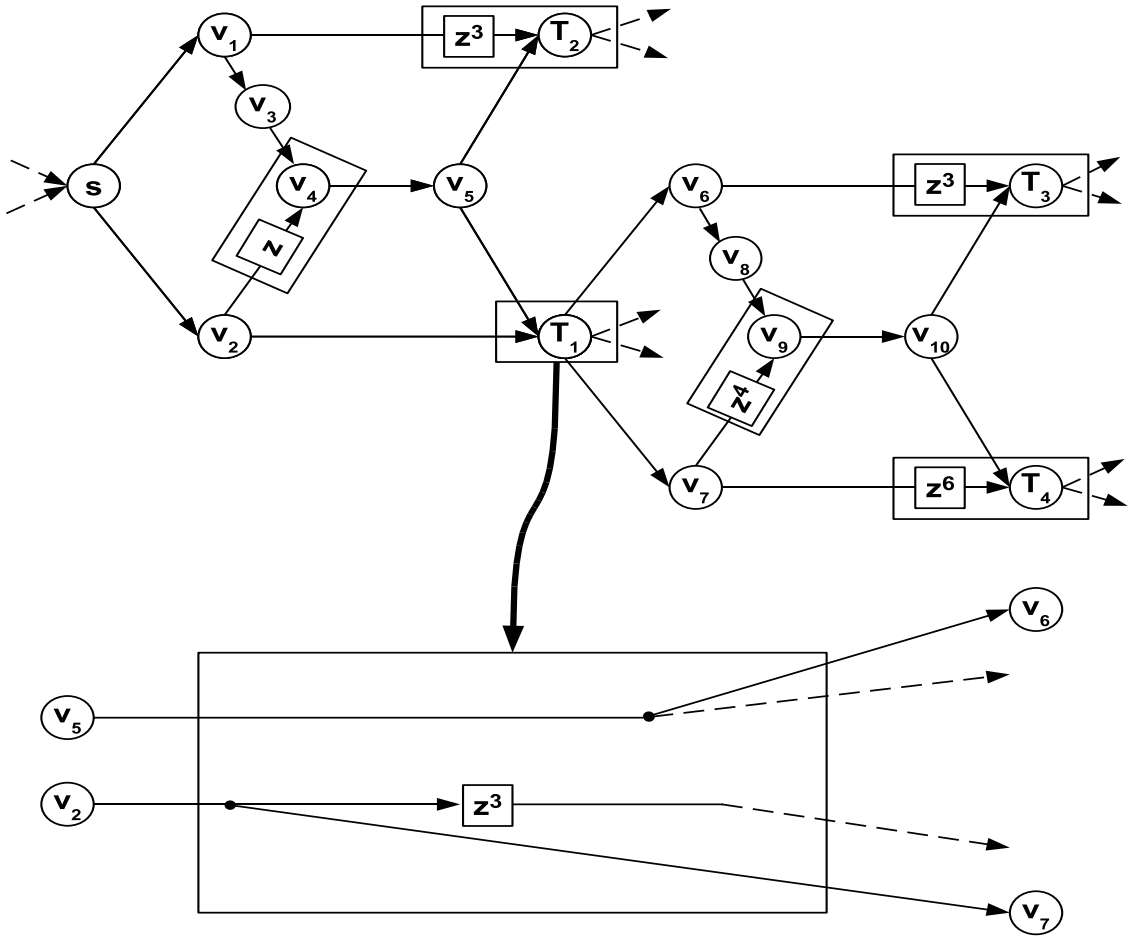

Fig. 10. Figure corresponding to Example 9. After line 10 of Algorithm 2, single-generation network coding has been implemented in the network and all the sinks see a network transfer matrix as in (1). Each box indicates the presence of memory elements at the associated node. The way sink T_1 uses memory is expanded below. Total memory used at this stage is 20.

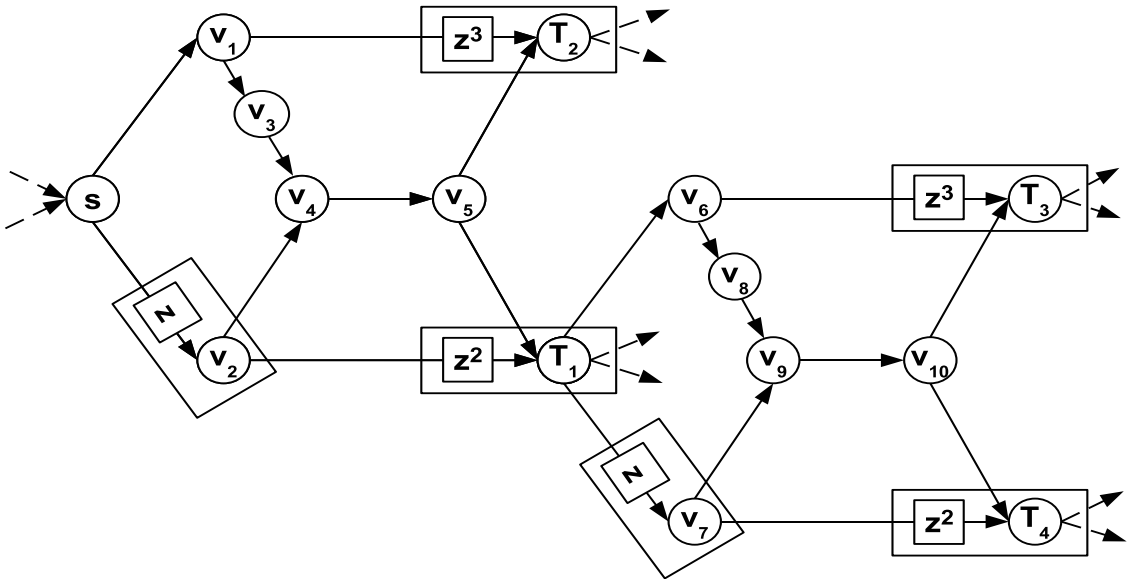

Fig. 11. Figure corresponding to Example 9. The network after line 24 of the algorithm. Comparing this figure with Fig. 10, memory reduction according to Subsubsection IV-B1 has resulted in the 'absorption' of memory elements from the nodes v_4, T_1, v_7, v_9, and T_4. Total memory used in the network now is 12.

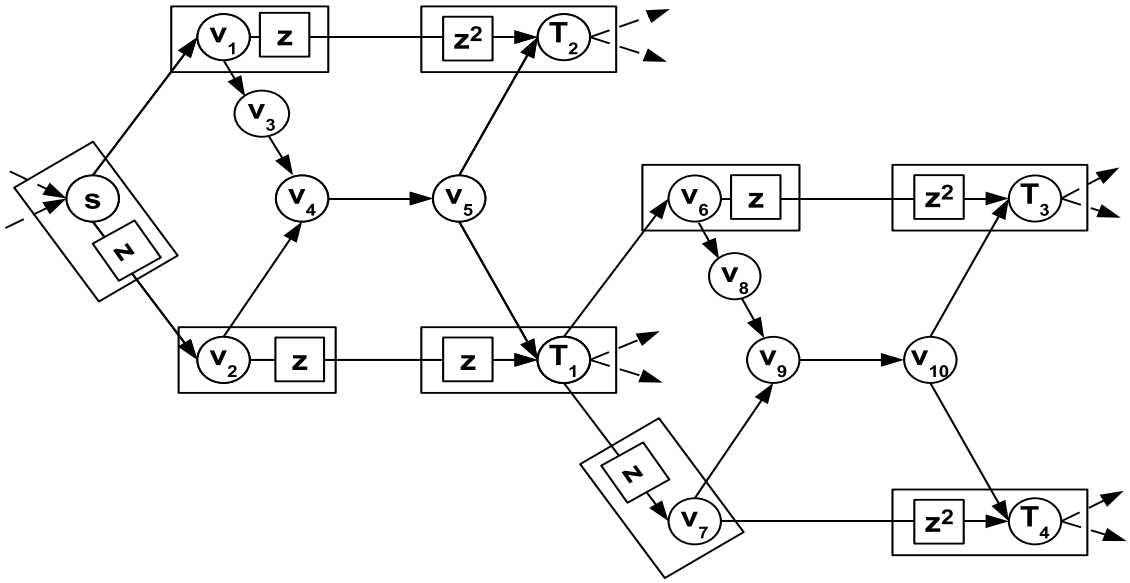

Fig. 12. Figure corresponding to Example 9. The network at the end of Algorithm 2. The 12 memory elements used in Fig. 11 are further distributed amongst the nodes of the network.

sinks use memory individually to decode, the total number of memory elements used in the network is 19, and all of them are used at the sinks. In the single-generation network coded network as shown in Fig. 12, it can be seen that the total number of memory elements used in the network is 12, out of which only 7 are used at the sinks, thereby showing a marked reduction from the memory-free case. The rest of the memory elements (numbering 5) are distributed across the nodes of the network.

C. Comparison with the approach of [5]

We can compare the straightforward approach of [5] and our approach to obtaining a single-generation network coded network for the modified unit-delay double-butterfly network of Fig. 9. According to the technique in [5], the result would be the network as in Fig. 10, thereby resulting in the use of 20 memory elements to obtain single-generation network coding. However, our algorithm utilizes the memory reduction and distribution techniques as given in Section IV and results in the output being as in Fig 12 using 12 memory elements and a more uniform distribution of memory elements across the network than in Fig. 10. Although the overall memory usage is reduced, it still remains to be shown whether Algorithm 2 actually obtains a configuration of the network with minimal number of memory elements being used to obtain single-generation network coding.

TABLE I
COMPARING INTER(MEMORY-FREE) AND SINGLE-GENERATION NETWORK CODING(USING MEMORY) FOR THE NETWORK IN FIG. 9

Sink	Network transfer matrix before Algorithm 2	Realizable decoding matrix obtained from $M_T^{-1}(z)$	Network transfer matrix after Algorithm 2	No. of memory elements used before Algorithm 2	No. of memory elements used after Algorithm 2
T_1	$M_{T_1}(z) = \begin{pmatrix} z & z^3 \\ 0 & z^4 \end{pmatrix}$	$P_{T_1}(z) = \begin{pmatrix} z^3 & z^2 \\ 0 & 1 \end{pmatrix}$	$M_{T_1}(z) = z^4 \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}$	3	1
T_2	$M_{T_2}(z) = \begin{pmatrix} z^3 & 0 \\ z^4 & z \end{pmatrix}$	$P_{T_2}(z) = \begin{pmatrix} 1 & 0 \\ z^3 & z^2 \end{pmatrix}$	$M_{T_2}(z) = z^4 \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}$	3	2
T_3	$M_{T_3}(z) = \begin{pmatrix} z^5+z^8 & z^5 \\ z^9 & z^6 \end{pmatrix}$	$P_{T_3}(z) = \begin{pmatrix} z & z^4 \\ 1 & 1+z^3 \end{pmatrix}$	$M_{T_3}(z) = z^9 \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}$	7	2
T_4	$M_{T_4}(z) = \begin{pmatrix} z^3 & z^5+z^8 \\ 0 & z^9 \end{pmatrix}$	$P_{T_4}(z) = \begin{pmatrix} z^6 & z^2+z^5 \\ 0 & 1 \end{pmatrix}$	$M_{T_4}(z) = z^9 \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}$	6	2

VI. IMPACT OF SINGLE-GENERATION NETWORK CODING ON NETWORK-ERROR CORRECTION

A. Impact on encoding

Construction of a CNECC: For details on the basics of convolutional codes, we refer the reader to [6]. The construction of a CNECC [4] for a given acyclic, unit-delay, memory-free network which corrects error vectors corresponding to a given set Φ of error patterns (an error pattern is a subset of $\mathcal{E}$ indicating the edges in error) can be summarized as follows

- Compute the set $\mathcal{W}_s$ of *error vector reflections* given by
$$\mathcal{W}_s = \bigcup_{T\in\mathcal{T},\rho\in\Phi} \left\{\boldsymbol{w}F_T(z)p_{_T}(z)M_T^{-1}(z) \mid \boldsymbol{w}\in\rho\right\}$$
where $\boldsymbol{w} \in F_q^{|\mathcal{E}|}$ is an error vector, and $\boldsymbol{w} \in \rho$ means that $\boldsymbol{w}$ matches an error pattern ρ. $p_{_T}(z) \in \mathbb{F}_q[z]$(the ring of polynomials) is some *processing function* chosen such that the *processing matrix* $p_{_T}(z)M_T^{-1}(z) = P_T(z)$ is a polynomial matrix.
- Let $t_s = \max_{\boldsymbol{w}_s(z)\in\mathcal{W}_s} w_H\left(\boldsymbol{w}_s(z)\right)$. Choose an input convolutional code $\mathcal{C}_s$ with free distance at least $2t_s+1$ as the CNECC for the given network.

The following lemma gives a bound on t_s and therefore the free distance demanded of the CNECC.

Lemma 2 ([4]): Given an acyclic, unit-delay, memory-free network $\mathcal{G}(\mathcal{V},\mathcal{E})$ with a given error pattern set Φ, let $T_{delay}-1$ be the maximum degree of any polynomial in the $F(z)$ matrix. Let w_H indicate the Hamming weight over $\mathbb{F}_q$. If r is the maximum number of non-zero coefficients of the polynomials $p_{_T}(z)$ corresponding to all sinks in $\mathcal{T}$, i.e $r = \max_{T\in\mathcal{T}} w_H\left(p_{_T}(z)\right)$, then we have

$$t_s \leq rn\left[(n+1)\left(T_{delay}-1\right)+1\right].$$

Algorithm 2 does not increase the value of T_{delay} in the matrix $F(z)$ because of the fact that an additional delay would not be introduced on any path between nodes which are at a distance of T_{delay} edges (the maximum number of edges on any path between any two nodes) from each other. Also, with memory being introduced in the nodes according to Algorithm 2, the network transfer matrices at all the sinks are of the form as given in (1). Therefore the processing functions at any sink T is of the form $p_{_T}(z) = z^{L_T}$, i.e $r = 1$.

Therefore we have that, for the network with delay and memory (used to achieve single-generation network coding),

$$t_s \leq n\left[(n+1)\left(T_{delay}-1\right)+1\right].$$

Thus, it is seen that the bound for t_s and therefore for the free distance demanded of the CNECC may be lower (if $r>1$) for the unit-delay, single-generation network coded network compared to the unit-delay, memory-free counterpart. However a decrease in the actual value of t_s cannot be guaranteed and has to be computed for every network individually in order to decide whether the CNECC designed for the unit-delay, memory-free network will continue to work for the single-generation network coded unit-delay counterpart.

B. Impact on decoding

Decoding of a CNECC: Let $G_I(z)$ be the generator matrix of the code $\mathcal{C}_s$ thus designed. Then we refer to the code $\mathcal{C}_s$ as the *input convolutional code* [3]. The effective code seen by a sink T is generated by the matrix $G_{O,T}(z) = G_T(z)M_T(z)$, which is known as the *output convolutional code* [3], $\mathcal{C}_{O,T}$, at sink T. The decoding of the CNECC at any sink T can be performed either on the trellis of the code $\mathcal{C}_s$ or that of the code $\mathcal{C}_{O,T}$ at that particular sink according to the free distance of $\mathcal{C}_{O,T}$ ($d_{free}(\mathcal{C}_{O,T})$), the catastrophic/non-catastrophic nature of $G_{O,T}(z)$, and a parameter called $T_{d_{free}}(\mathcal{C}_{O,T})$, whose definition for a rate b/c code $\mathcal{C}$ over $\mathbb{F}_q$ is given in [3] as follows.

$$T_{d_{free}}(\mathcal{C}) := \max_{\boldsymbol{v}_{[0,j)}\in S_{d_{free}}} j+1 \tag{26}$$

where $S_{d_{free}}$ [3] is defined as follows.

$$S_{d_{free}} := \left\{\boldsymbol{v}_{[0,j)} \mid w_H\left(\boldsymbol{v}_{[0,j)}\right) < d_{free}(\mathcal{C}), \boldsymbol{\sigma}_0 = \boldsymbol{0}, \forall\ j>0\right\}$$

where

$$\boldsymbol{v}_{[0,j)} := [\boldsymbol{v}_0, \boldsymbol{v}_1, ..., \boldsymbol{v}_{j-1}]$$

is a truncated codeword sequence with $\boldsymbol{v}_i \in \mathbb{F}_q^c$), $\boldsymbol{\sigma}_t$ indicates the content of the delay elements in the encoder at a time t, and w_H indicates the Hamming weight over $\mathbb{F}_q$. The set $S_{d_{free}}$ consisting of all possible truncated codeword sequences $\boldsymbol{v}_{[0,j)}$ of weight less than $d_{free}(\mathcal{C})$ that start in the zero state. Then, we have the following proposition.

Proposition 1 ([3]): The minimum Hamming weight trellis decoding algorithm can correct all error sequences which have the property that the Hamming weight of the error sequence in any consecutive $T_{d_{free}}(\mathcal{C})$ segments (a segment being a collection of c output symbols corresponding to every b input symbols) is utmost $\left\lfloor \frac{d_{free}(\mathcal{C})-1}{2} \right\rfloor$.

With the CNECC in place in a unit-delay. memory-free network, under certain conditions (see Subsection IV-D of

[4]), a sink has to decode on the trellis of the input convolutional code, in which case the sink has to multiply the incoming n output streams with the processing matrix $P_T(z)$, which may require additional memory elements to implement. However, with a single-generation network code implemented using memory elements, part of this processing is done in a distributed manner in the other nodes of the network, thereby decreasing the memory requirement at the sinks.

In the forthcoming section, we further observe the advantages that the use of memory in the intermediate nodes offers in the performance of CNECCs under a probabilistic error setting.

VII. Simulation results

A. A probabilistic error model

Probabilistic error models have been considered in the context of random network coding in [7]. We define a probabilistic error model for a unit delay network $\mathcal{G}(\mathcal{V}, \mathcal{E})$ by defining the probabilities of any set of i $(i \leq |\mathcal{E}|)$ edges of the network being in error at any given time instant. Across time instants, we assume that the network errors are i.i.d. according to this distribution.

$$Prob.(i \text{ network edges being in error}) = p^i \tag{27}$$

$$Prob.(\text{no edges are in error}) = q \tag{28}$$

where $1 < i \leq |\mathcal{E}|$, and $p, q \leq 1$ are real numbers indicating the probability of any single edge error in the network and the probability of no edges in error respectively, such that $q + \sum_{i=1}^{|\mathcal{E}|} p^i = 1$.

B. Simulations on the modified butterfly network

Fig. 13 on the top of the next page shows a modified butterfly network before and after running Algorithm 2. This network is clearly a part of the modified double-butterfly network of Fig. 9, and the associated matrices at the sinks T_1 and T_2 are given in Table I. With the probability model as in (27) and (28) with $|\mathcal{E}| = 10$ for this network, we simulate the performance of 3 input convolutional codes implemented on this network for both the with-memory and memory-free cases as in Fig. 13 with the sinks performing hard decision decoding on the trellis of the input convolutional code.

In the following discussion we refer to sinks T_1 and T_2 of Fig. 13 as Sink 1 and Sink 2. The 3 input convolutional codes and the rationality behind choosing them are given as follows.

- Code $\mathcal{C}_1$ is generated by the generator matrix

$$G_{I_1}(z) = \begin{bmatrix} 1+z & 1 \end{bmatrix},$$

  with $d_{free}(\mathcal{C}_1) = 3$ and $T_{d_{free}}(\mathcal{C}_1) = 2$. This code is chosen only to illustrate the error correcting capability of codes with low values of $d_{free}(\mathcal{C})$ and $T_{d_{free}}(\mathcal{C})$.
- Code $\mathcal{C}_2$ is generated by the generator matrix

$$G_{I_2}(z) = \begin{bmatrix} 1+z^2 & 1+z+z^2 \end{bmatrix},$$

  with $d_{free}(\mathcal{C}_2) = 5$ and $T_{d_{free}}(\mathcal{C}_2) = 6$. This code corrects all double edge errors in the instantaneous version (with all edge delays and memories being zero) of Fig. 13 as long as they are separated by 6 network uses.
- Code $\mathcal{C}_3$ is generated by the generator matrix

$$G_{I_3}(z) = \begin{bmatrix} 1+z+z^4 & 1+z^2+z^3+z^4 \end{bmatrix},$$

  with $d_{free}(\mathcal{C}_3) = 7$ and $T_{d_{free}}(\mathcal{C}_3) = 12$. This code corrects all double edge errors in the unit-delay network given in Fig. 13 as long as they are separated by 12 network uses.

We note here that values of $T_{d_{free}}(\mathcal{C})$ of the 3 codes are directly proportional to their free distances, i.e, the code with greater free distance has higher $T_{d_{free}}(\mathcal{C})$.

Fig. 14 and Fig. 15 illustrate the BERs for these 3 codes for both the with-memory and memory-free case for different values of the parameter p (the probability of a single edge error) of (27). Clearly the BER values fall with decreasing p.

The description and explanation of the regions marked 'd_{free} dominated region' and '$T_{d_{free}}$ dominated region' (named so according to the dominant parameter in those regions) are given in [3]. In the following discussion, we concentrate on the comparison between the performance of every code in the memory-free and the with-memory case. Towards that end, we recall from Proposition 1 that both the Hamming weight of error events and the separation between any two consecutive error events are important to correct them.

Performance improvement of CNECCs with memory at the intermediate nodes:

1) With respect to codes $\mathcal{C}_2$ and $\mathcal{C}_3$, we see that there is an improvement in performance when memory is used at the intermediate nodes. This is because of the fact that the presence of memory elements in the network results in a clumping-together of error bits at the sinks. For example, assume that in the network of Fig. 13, an error occurs in edge $s \rightarrow v_1$ at time instant t_1. We consider the situation at Sink 2. In the memory-free case, the effect of this error is felt at different time instants at the two incoming edges of Sink 2, at $t_1 + 1$ and at $t_1 + 4$. However, with memory elements at the intermediate nodes, the effects of the edge error now occur at the same time instant $(t_1 + 4)$ in both the incoming edges of Sink 2. The effect of such errors cumulatively result in more error events (with less Hamming weights each) in the memory-free case (because of the distribution of errors) and less error events (with comparatively more Hamming weights each) in the with-memory case (as a result of clumped errors). However, because Codes $\mathcal{C}_2$ and $\mathcal{C}_3$ have enough free distance, the number of such error events is what dominates the performance. Therefore Codes $\mathcal{C}_2$ and $\mathcal{C}_3$ correct more errors in the with-memory case. The same effect may be observed at Sink 1 also.
2) With respect to the code $\mathcal{C}_1$, there is no observable change in performance between the memory-free and with-memory cases. We note that the same effect is observed with the errors as in the previous case. But because of $T_{d_{free}}(\mathcal{C}_1)$ being less (only 2), the clumping together of error bits does not benefit much. Therefore there is no significant improvement in performance.

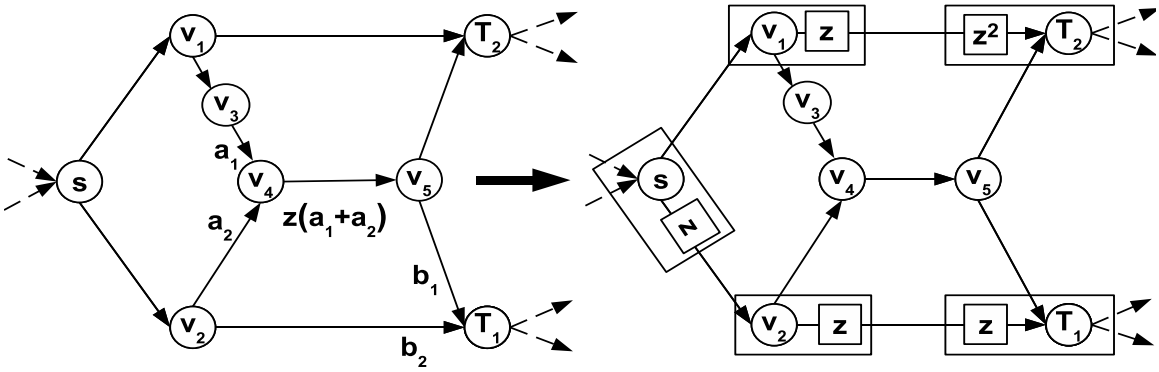

Fig. 13. A modified butterfly network

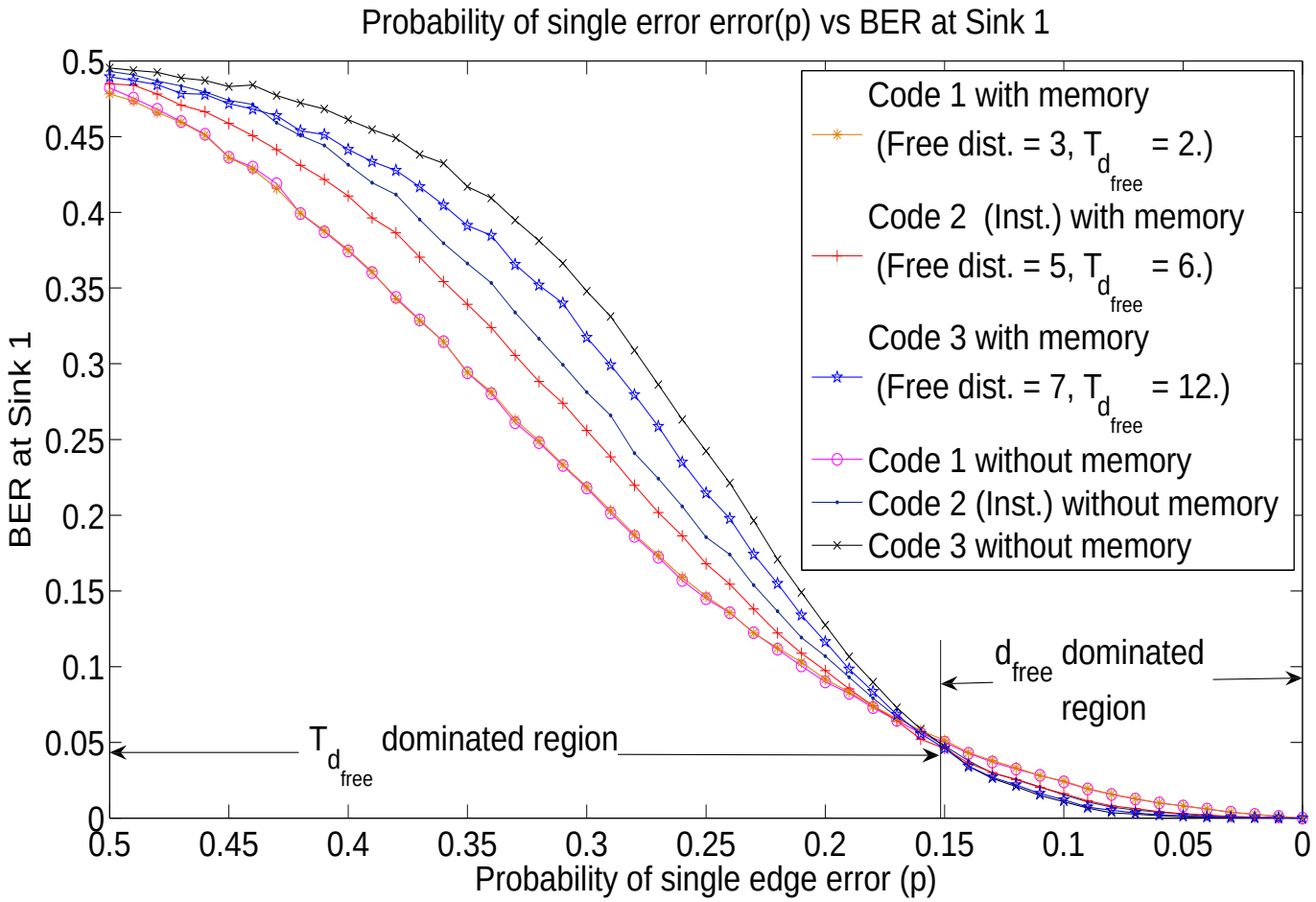

Fig. 14. BER (with and without memory) at Sink 1

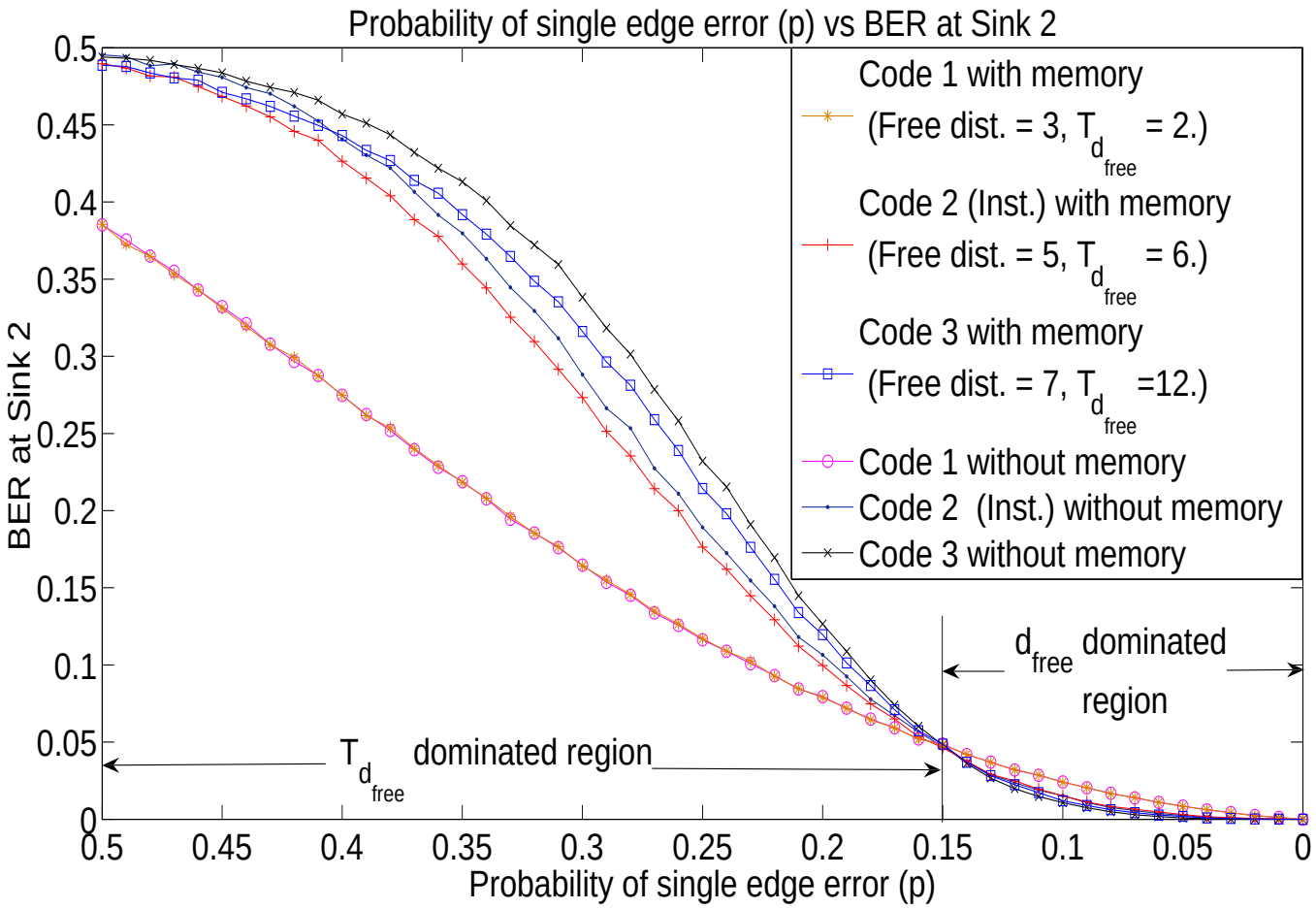

Fig. 15. BER (with and without memory) at Sink 2

3) There is no significant difference in the performance of any code between the memory-free and the with-memory case in the 'd_{free} dominated region.' This is because of the fact that the errors that occur in the network are already sparse.

ACKNOWLEDGMENT

This work was supported partly by the DRDO-IISc program on Advanced Research in Mathematical Engineering through a research grant to B. S. Rajan.

REFERENCES

[1] R. Ahlswede, N. Cai, R. Li and R. Yeung, "Network Information Flow", IEEE Transactions on Information Theory, vol.46, no.4, July 2000, pp. 1204-1216.

[2] R. Koetter and M. Medard, "An Algebraic Approach to Network Coding", IEEE/ACM Transactions on Networking, vol. 11, no. 5, Oct. 2003, pp. 782-795.

[3] K. Prasad and B. Sundar Rajan, "Convolutional codes for Network-error correction", arXiv:0902.4177v3 [cs.IT], August 2009, Available at: http://arxiv.org/abs/0902.4177. A shortened version of this paper is to appear in the proceedings of Globecom 2009, Nov. 30 - Dec. 4, Honolulu, Hawaii, USA.

[4] K. Prasad and B. Sundar Rajan, "Network error correction for unit-delay, memory-free networks using convolutional codes", arXiv:0903.1967v3[cs.IT], September 2009, Available at: http://arxiv.org/abs/0903.1967.

[5] X. Wu, C. Zhao and X. You, "Generation-Based Network Coding over Networks with Delay", IFIP International Conference on Network and Parallel Computing, Shangai, China, Oct. 18-21 2008, pp. 365-368.

[6] R. Johannesson and K.S Zigangirov, Fundamentals of Convolutional Coding, John Wiley, 1999.

[7] D. Silva,F. R Kschischang, and R. Koetter, "Capacity of random network coding under a probabilistic error model", 24th Biennial Symposium on Communications, Kingston, USA, 24-26 June 2008, pp. 9-12.